Rob Bickel Agency · Managed Software Services

The Orphaned Software Playbook

Restoring ownership of the applications your business runs on. An enterprise architecture approach for established mid-sized companies.

For COOs, CFOs, CIOs, and operations leaders

robbickel.com

2026 edition

In brief

Most established companies run at least one business-critical application that nobody is accountable for. It was built by a contractor, an employee who has since moved on, or a vendor engagement that closed. It still runs billing, scheduling, fulfilment, or onboarding. It mostly works, and people are reluctant to touch it.

This playbook treats that situation as what it is: a gap in ownership, not a failure of technology. It draws on enterprise architecture practice, on the published research on key-person risk and software cost, and on the governance frameworks your auditors and customers already recognize.

The research is consistent. In a study of 1,932 open-source projects, the projects that lost their key developers survived only when someone new took ownership. Nothing else predicted recovery.[1]

Most of software's cost arrives after launch. Robert Glass's widely cited synthesis puts maintenance at 40 to 80 percent of lifetime cost, and most of that is the system keeping pace with the business.[2]

AI is accelerating the problem. Employees now build working tools in an afternoon, and the AI models those tools depend on have published retirement dates.[3,4]

Ownership can be restored without a rewrite. The seven-step sequence in Part 8 restores control of an orphaned application in weeks, not quarters.

1. What an orphan is, and why it is a capability problem

An orphaned application is software the business depends on with no one accountable for keeping it healthy, changing it safely, or retiring it on purpose. It is rarely abandoned in any dramatic sense. It simply fell out of every process that would have kept it owned.

Enterprise architecture gives a more precise way to describe this. TOGAF, the most widely used architecture framework, starts from **business capabilities**: the things the business must be able to do, such as Order Fulfilment, Customer Onboarding, or Billing. Applications are the means by which capabilities are realized. TOGAF's Application Portfolio Catalog exists "to identify and maintain a list of all the applications in the enterprise," along with what each one supports.[5]

Seen that way, an orphaned application is an **unowned capability**. The business owner can describe what the capability does. Nobody can say who maintains the systems behind it. That distinction matters because it changes the remedy. An application problem invites a technical fix, often a rewrite. A capability problem is closed with governance: a named owner, a known lifecycle state, and a level of service that matches the capability's importance.

The stakes are large because established organizations spend most of their technology money running what already exists. The US Government Accountability Office reports that federal agencies typically spend about 80 percent of their IT budgets, out of more than $100 billion a year, operating and maintaining existing systems.[6] Public-sector budgets are not a perfect proxy for a 300-person company, but they are the best freely citable view of where an established organization's technology spend actually goes: overwhelmingly into what is already running.

The first test of ownership is therefore simple. **Can your company produce a list of its applications, with a named business owner and a named technical owner against each one?** Most mid-sized companies cannot, and the gaps in that list are where the orphans are.

2. Key-person risk: the truck factor

Software engineers have a blunt name for concentration of knowledge: the *truck factor*, the number of people who would have to leave before a project stalls. It has been studied carefully.

- A 2016 study of 133 popular open-source projects found that **65 percent had a truck factor of two or fewer**. In most, the departure of one or two people would leave nobody able to maintain the code.[7]
- A follow-up study tracked 1,932 projects. **315 of them (16 percent) lost their truck-factor developers. Only 128 of those, or 41 percent, survived**, and every survivor did so because new core developers took over.[1]
- A 2024 replication across more than 36,000 projects found that **89 percent had lost their core team at least once**, and that **only 27 percent of abandoned projects attracted a new key developer**.[8]

Two honest caveats apply. These are open-source projects, which typically have far larger contributor pools than an internal tool at a mid-sized company. The internal picture is plausibly worse, but nobody has measured it. And the research describes code, not the business rules and vendor relationships that also live in people's heads.

The time dimension makes the risk structural rather than unlucky. The US Bureau of Labor Statistics reports that the **median tenure of workers in computer and mathematical occupations was 4.6 years in January 2026**.[9] A

business-critical application is usually expected to run for a decade or more. On those numbers, a system will typically outlive its builder's time at the company, and planning as though it will not is a decision in itself.

The lesson from the research is the thesis of this playbook. **Projects recovered from losing their key people only when someone new took ownership.** Documentation helped. Monitoring helped. But the deciding factor was an accountable successor.

3. Knowledge loss and the cost of re-learning

When the builder leaves, the knowledge does not disappear evenly. What usually survives is the code. What is usually lost is the reasoning: why a rule exists, which customer it was written for, which downstream process quietly depends on a nightly job.

Re-learning is expensive even under good conditions. A study published in *IEEE Transactions on Software Engineering* followed 78 professional developers across seven projects and 3,148 working hours. They spent **about 58 percent of their time on program comprehension**, simply understanding code, and junior developers spent a larger share than seniors.[10] That was on code they worked with every day. For an undocumented system nobody has touched in two years, the share is plausibly higher. The seniority gap is also practical guidance: inherited systems are where experienced engineers pay for themselves.

Two operational findings point to what reduces the cost:

- **Documentation quality pays off measurably.** Google's DORA research program reported in 2023 that teams with high-quality documentation showed 25 percent higher team performance, and that reliability practices had markedly more impact where documentation was strong.[11]

- **Written response procedures shorten recovery.** Google's *Site Reliability Engineering* book observes that a prepared playbook "produces roughly a 3x improvement in MTTR as compared to the strategy of 'winging it.'"[12] That is Google's stated experience rather than a controlled study, but it matches what most operations leaders see.

A useful test. Could a competent engineer who has never seen this system deploy a safe fix using only what is written down? If the honest answer is no, the system's continuity depends on a person rather than on the company.

4. Access custody: who holds the keys

The quietest form of orphaning is administrative. The hosting account was opened on a contractor's card. The domain is registered to a former employee's email address. The API keys sit in a laptop's configuration file. Everything works until someone needs to change it, renew it, or prove who has access to it.

NIST's Cybersecurity Framework 2.0 states the expected outcome plainly: "Identities and credentials for authorized users, services, and hardware are managed **by the organization**."[13] The last three words are the custody test. Credentials held personally by builders, however trustworthy, are not managed by the organization.

The external data shows why this matters for continuity. Verizon's 2025 Data Breach Investigations Report analysed 12,195 confirmed breaches and found that **the abuse of legitimate credentials was the most common way in, at 22 percent** of breaches where the entry point was known. **Involvement of a third party doubled, from 15 to 30 percent.**[14] Secrets also outlive the people who

created them: a secrets-detection vendor reports that 70 percent of credentials it found leaked in 2022 were still valid when re-tested in 2025.[15] The failure is rarely detection. It is that nobody owns revocation.

Restoring custody is unglamorous and decisive:

- Move every administrative account, domain, repository, cloud subscription, and vendor portal to company-owned identities and company billing.
- Store credentials in a managed vault, rotate anything a former builder could have seen, and record who can do what.
- Make revocation part of every departure, including contractors and service accounts, on the day of departure.

5. The real cost of software after launch

Leadership teams often treat an orphaned system as cheap because nobody is spending money on it. The research suggests the opposite: the money is being deferred, not saved.

Robert Glass's *Facts and Fallacies of Software Engineering*, one of the most cited syntheses in the field, puts **maintenance at 40 to 80 percent of total software cost, averaging about 60 percent**. Of that, roughly **60 percent is enhancement**, the system being changed to keep pace with the business, and only around 17 percent is correcting errors.[2] For a CFO, the reframing is important. Most post-launch spending is not repair. It is the price of a system staying useful. An orphan with no budget for change is not avoiding that cost. It is accumulating it as risk.

Three further sources describe the scale of that accumulation, each with a caveat worth stating:

- **McKinsey's 2020 survey of 50 CIOs** at companies with more than $1 billion in revenue found that they estimated technical debt at **20 to 40 percent of**

the value of their entire technology estate, and that less than 20 percent of technology budgets went to paying it down.[16] These are self-estimates by large firms, but they are the clearest board-level figure available.

- **Stripe's 2018 *Developer Coefficient*** survey reported developers spending about **13.5 hours a week on technical debt and 3.8 hours on bad code**, out of a 41.1-hour week.[17]
- **The Consortium for Information and Software Quality** estimated accumulated US technical debt at about **$1.52 trillion** in 2022.[18] It is a macro estimate, useful only for scale.

Gartner has published a research note titled "Technical Debt Benchmarks Are a Waste of Time."[19] The point is well made. What matters is not how your estate compares to an industry average, but whether you know your own baseline: what each capability costs to run, what it would cost to change, and what you are deferring.

6. Handover done right

Most orphans are created at a single moment: the end of a build. The project closes, the team disperses, and nothing carries the solution into ongoing ownership.

TOGAF describes exactly the step that is usually skipped. Its implementation governance phase (Phase G) "establishes the connection between architecture and implementation organization, through the **Architecture Contract**," and a signed contract is one of its outputs.[20] The following phase, architecture change management (Phase H), keeps the delivered solution aligned with the business as both change. An orphan, in these terms, is a solution that left the build without an Architecture Contract and never entered the change-management loop.

Mid-sized companies do not need TOGAF's full apparatus to close that gap. They need its intent, written into a short handover agreement that names:

- **The accountable owner** after go-live, business and technical, and the service level each capability warrants.
- **What is handed over:** source code, infrastructure definitions, credentials, runbooks, architecture notes, and open issues, all in company custody.
- **How change will be made:** who assesses, approves, tests, and releases changes. ITIL 4 describes the purpose of change enablement as maximizing "the number of successful service and product changes by properly assessing risks, authorizing changes to proceed, and managing the change schedule."[21]
- **The exit path:** what happens if the supplier relationship ends.

Government digital services have codified the same lessons. The US 18F *De-risking Government Technology* guide added a vendor-management section in its 2024 revision,[22] and the Canadian Digital Service's technical playbook directs teams to avoid vendor lock-in.[23] The operational data points the same way. Uptime Institute's 2025 outage analysis found that IT and networking causes rose to 23 percent of impactful outages, which it associates with change management and misconfiguration.[24] Outages increasingly come from how systems are run. That is precisely what an orphan lacks.

7. AI-built tools: orphaned from birth

Every pattern above is being compressed by AI. A capable employee can now produce a working integration in an afternoon and an agent that automates a workflow in a week. The speed is genuinely valuable. It also means realizations of business capabilities are being created far faster than ownership is being assigned.

Adoption is running well ahead of governance:

- Microsoft and LinkedIn's 2024 Work Trend Index (31,000 people in 31 countries) found that **75 percent of knowledge workers use generative AI at work, and 78 percent of those users bring their own AI tools**.[3]
- Stack Overflow's 2025 developer survey (more than 49,000 responses) found **84 percent of developers using or planning to use AI tools, while 46 percent do not trust the accuracy of the output**, up from 31 percent a year earlier.[25]
- IBM's 2025 *Cost of a Data Breach* study (600 organizations) found that **one in five organizations reported a breach involving shadow AI**, and that **63 percent of breached organizations had no AI governance policy or were still developing one**.[26]

The quality evidence argues for review, not alarm. In a Stanford study presented at ACM CCS 2023, participants using an AI assistant wrote less secure code in four of five tasks, **and were more likely to believe their code was secure**.[27] Veracode's 2025 benchmark found security flaws in 45 percent of AI-generated solutions across more than 100 models, on tasks designed to probe known weaknesses.[28] DORA's research describes AI as an "amplifier" of whatever practices an organization already has.[29]

The most distinctive point is also the least alarming. **An AI feature built on a vendor model has a published expiry date.** Anthropic commits to "at least 60 days' notice" before retiring a model, and retired Claude 3.5 Sonnet in October 2025, about ten weeks after announcing its deprecation. OpenAI commits to at least six months for generally available models, and as little as two weeks for previews.[4] An AI component nobody owns does not degrade gracefully. It stops on a known date.

The NIST AI Risk Management Framework sets out the governance response in terms most auditors will recognize. GOVERN 1.6 calls for mechanisms "to inventory AI systems." GOVERN 2.1 calls for documented roles and responsibilities. GOVERN 1.7 calls for processes to decommission AI systems safely. MANAGE 3.2 requires that pre-trained models "are monitored as part of AI system regular monitoring and maintenance."[30] ISO/IEC 42001 builds a management system around the same expectations, including controls for the AI system life cycle.[31] For a mid-sized company, the practical version is simple: **every AI component goes on the application list, gets a technical owner, and gets a lifecycle state, exactly like any other software.**

8. The restoration sequence

Ownership can be restored without rebuilding anything. The sequence below is the one I use with clients. Steps 1 to 4 typically take a few weeks for a single capability. Steps 5 to 7 then become the ongoing rhythm.

> **1. Inventory against capabilities.** Build the Application Portfolio Catalog: every application, integration, scheduled job, spreadsheet with business rules, and AI component, mapped to the business capability it serves. Interview by capability, not by system. Asking the owner of Customer Onboarding to walk through a new customer surfaces the script and the AI-assisted email workflow that a system-by-system interview misses. Confirm the list against twelve months of payments, single sign-on records, and cloud accounts.

> **2. Name two owners.** For each capability, a business owner who decides what it must do, and a technical owner accountable for every component that realizes it, whoever built them.

> **3. Take custody.** Move accounts, domains, repositories, and billing to company identities, vault the credentials, and rotate anything former builders could reach.

> **4. Baseline and document.** Describe the components, their dependencies, and their data flows. Write the runbook for the three most likely failures. A simple diagram of the capability, its applications, and their interfaces is enough.

> **5. Stabilize.** Add monitoring that alerts a named person, confirm backups run, and **perform a test restore**. Bring the runtime and dependencies to supported versions, or record a deliberate exception.

> **6. Decide with TIME.** Classify each component using Gartner's TIME model, as widely described: **Tolerate, Invest, Migrate, or Eliminate**, based on business value and technical fit.[32] LeanIX describes Tolerate as "more a stay of execution pending appeal." That is a useful warning. An orphan is usually a Tolerate that nobody is reviewing.

> **7. Enter the change loop.** Put the capability on a regular cadence: changes assessed against the capability map, a quarterly review of cost, risk, and lifecycle state, and the delivery measures DORA recommends, such as change fail rate and failed-deployment recovery time.[33]

The US federal government's own legacy-systems review offers a useful benchmark for discipline. In 2025, the GAO examined 69 legacy systems, flagged 11 as most in need of modernization, and found that only three agencies had documented modernization plans covering all key practices.[34] Having a plan, owned and current, is itself the differentiator.

9. Who owns it next

The final decision is who carries the ownership forward. There are three sound answers, and many companies use a mix.

- **An internal owner** works well when the capability is core to how the company competes, the technology is mainstream, and the company can retain at least two people who understand it.
- **An outside partner** works well when the capability is essential but not distinctive, when the skills required are specialized or intermittent, or when the company needs coverage beyond what one or two employees can provide.
- **A hybrid** keeps business ownership and product decisions inside the company, while a partner holds technical ownership, change management, and response.

Whichever you choose, the commitment should be written down: the owner, the scope, the service level for each capability, how changes are approved, and how the relationship ends. That is the Architecture Contract, scaled to a mid-sized company. Without it, the next departure creates the next orphan.

Where my team fits

This is the work Rob Bickel Agency does. We take **after-build ownership** of the applications behind a set of business capabilities: monitored, patched, recoverable, and documented, at a service level matched to each one's importance. We own **change management** for them, so every change is assessed, tested, released, and recorded. And when something critical fails and the people who built it are gone, our **break-fix and urgent consulting** work stabilizes it first, diagnoses the real cause, and leaves the system in company custody, documented, and on the path to ongoing ownership.

Many client relationships start with that urgent call. The rest start with a portfolio audit that builds the capability map and tells leadership what is unowned.

Sources and notes

Figures are quoted as published. Vendor surveys are labelled; they are useful for direction, and their publishers sell related products. TOGAF quotations are from The Open Group's public TOGAF 9 pages; current TOGAF Standard text paraphrased.

1. Avelino, Constantinou, Valente, Serebrenik, "On the abandonment and survival of open source projects," ESEM 2019. arxiv.org/abs/1906.08058
2. Robert L. Glass, *Facts and Fallacies of Software Engineering*, Addison-Wesley, 2002, facts on maintenance.
3. Microsoft and LinkedIn, *2024 Work Trend Index*. microsoft.com
4. Anthropic, *Model deprecations*; OpenAI, *Deprecations*. platform.claude.com; developers.openai.com
5. The Open Group, TOGAF 9, *Application Portfolio Catalog*. pubs.opengroup.org
6. US GAO, GAO-23-106821 and GAO-21-524T, federal IT operations and maintenance spending. gao.gov
7. Avelino, Passos, Hora, Valente, "A novel approach for estimating truck factors," ICPC 2016. arxiv.org/abs/1604.06766
8. Large-scale replication of truck-factor and project-abandonment research, 2024. arXiv 2412.00313
9. US Bureau of Labor Statistics, *Employee Tenure*, January 2026, Table 6. bls.gov
10. Xia, Bao, Lo, Xing, Hassan, Li, "Measuring program comprehension: a large-scale field study with professionals," *IEEE Transactions on Software Engineering*, 2018.
11. Google Cloud, *2023 Accelerate State of DevOps Report* announcement. cloud.google.com
12. Beyer et al., *Site Reliability Engineering*, Google, Introduction. sre.google
13. NIST, *Cybersecurity Framework 2.0* (CSWP 29), February 2024, PR.AA-01. nist.gov
14. Verizon, *2025 Data Breach Investigations Report*. verizon.com
15. GitGuardian, *The State of Secrets Sprawl 2025* (vendor). gitguardian.com
16. McKinsey & Company, "Tech debt: Reclaiming tech equity," 2020. mckinsey.com
17. Stripe, *The Developer Coefficient*, 2018.
18. CISQ, *The Cost of Poor Software Quality in the US: A 2022 Report*. it-cisq.org
19. Gartner, "Technical Debt Benchmarks Are a Waste of Time" (research note).
20. The Open Group, TOGAF 9, Phase G: Implementation Governance. pubs.opengroup.org
21. ITIL 4 change enablement purpose, as published in AWS, *Change Enablement in the Cloud*. docs.aws.amazon.com
22. 18F / GSA, *De-risking Government Technology*, Guide 2.0, September 2024. digital.gov
23. Canadian Digital Service, *Technical Playbook*. cds-snc.github.io
24. Uptime Institute, *Annual Outage Analysis 2025*, press release. uptimeinstitute.com
25. Stack Overflow, *2025 Developer Survey*. stackoverflow.co

26. IBM and Ponemon Institute, *Cost of a Data Breach Report 2025*. ibm.com
27. Perry, Srivastava, Kumar, Boneh, "Do users write more insecure code with AI assistants?", ACM CCS 2023. arxiv.org/abs/2211.03622
28. Veracode, *2025 GenAI Code Security Report* (vendor). veracode.com
29. Google, *2025 DORA State of AI-assisted Software Development*. dora.dev
30. NIST, *AI Risk Management Framework 1.0* (AI 100-1), Core. airc.nist.gov
31. ISO/IEC 42001:2023, *Artificial intelligence: Management system*, Annex A. iso.org
32. LeanIX, "Gartner TIME model" and "TIME: Tolerate." leanix.net
33. DORA, *DORA's software delivery metrics*. dora.dev
34. US GAO, GAO-25-107795, review of critical federal legacy IT systems, July 2025. gao.gov

Rob Bickel Agency · Managed Software Services

Need an owner for the software your business runs on?

After-build ownership, change management, and urgent break-fix for the applications established, mid-sized companies depend on. I lead every engagement personally, supported by our onshore and nearshore engineers.

bickel.rob@gmail.com

robbickel.com